## CS 499 Milestone One: Code Review Script

## **Introduction**

Hey Class, my name is Joseph Klenk. This is my code review for CS 499 Milestone One. I'll review three artifacts from my previous coursework that I'll enhance for my ePortfolio, covering:

1. Software Design and Engineering
2. Algorithms and Data Structures
3. Databases

## **Category One: Software Engineering and Design**

### **Existing Functionality**

* Android mobile application from CS 360
* Features:
  + Login system with authentication
  + SQLite database for user credentials
  + Inventory management with grid display
  + CRUD operations for inventory items
  + SMS notifications for low inventory
  + Android permissions handling

This Android application includes inventory management with user authentication capability. Users have an account and log in securely through a SQLite database as a credential repository. The application enables users to display inventory entries in a grid view and execute all common database actions: create, read, update, and delete. The application also includes SMS notifications when inventory quantity drops below a set threshold level, with correct Android runtime permission management for SMS support.

### **Code Analysis**

* **Structure Issues**: No clear architecture pattern; UI, database, and business logic are mixed
* **Error Handling**: Minimal exception handling in database operations
* **Security Concerns**: Plaintext password storage in SQLite
* **Performance**: Loads all inventory items at once with no pagination
* **Maintainability**: Lacks comments and has inconsistent naming conventions

Based on going through the code, I've found a number of serious issues that must be addressed. The application does not have an appropriate design pattern as UI elements, database queries, and business logic are all coupled into the same classes. Error handling is sparse throughout the application codebase, especially in performing database queries, and hence might result in application failures. From a security point of view, user passwords are being stored in plaintext using the SQLite database, making it a major risk factor. Performance issues emanate from loading all inventory all at once and not using pagination and maintaining it, while overall maintainability is hampered by poor documentation and random naming conventions.

### **Enhancement Plans**

* Implement MVC architecture to separate concerns
* Add password hashing for improved security
* Implement proper exception handling
* Add pagination for inventory display
* Improve code documentation and naming consistency

My plan is to apply the Model-View-Controller design pattern to keep concerns separate and maintainability high. BCrypt password hashing is to be added to safely store user passwords and avoid unauthorized access. Robust exception handling is to be included in all places in the application, especially for database queries, to enhance stability. For performance issues, pagination is to be added to display inventory so that bigger datasets can be handled by the application more effectively. Last but not least, code documentation through JavaDoc comments is to be enhanced and naming conventions standardized throughout the application. These improvements map up to learning objectives 3, 4, and 5 by adopting secure and well-designed solutions that comply with industry standards.

## **Category Two: Algorithms and Data Structures**

### **Existing Functionality**

* Pirate Intelligent Agent from CS 370
* Features:
  + Deep Q-learning algorithm for pathfinding
  + Epsilon-greedy exploration approach
  + Neural network for action prediction
  + Training loop with reward-based updates
  + Basic performance evaluation

This is a Pirate Intelligent Agent designed for a game of treasure hunt with reinforcement learning principles. The agent follows a deep Q-learning method to move through a maze-based game world and reach treasure ahead of a human opponent. An epsilon-greedy policy is used to achieve a tradeoff among learning and exploration throughout learning. The neural network model makes predictions for optimal actions given the current game state and the loop for training continuously adjusts the Q-network based upon received rewards. There is added support with simple functions to monitor the performance of the agent within the game world.

### **Code Analysis**

* **Algorithm Efficiency**: Random exploration slows convergence
* **Data Structures**: Simple list for experience replay lacks prioritization
* **Code Clarity**: Many magic numbers and limited comments
* **Modularity**: Poor separation of concerns between components
* **Testing**: Limited evaluation across different scenarios

My analysis reveals several opportunities for improvement in this reinforcement learning implementation. The current epsilon-greedy approach uses random exploration, which significantly slows convergence and requires excessive training episodes. The experience replay mechanism utilizes a simple list structure that fails to prioritize important learning experiences. Code clarity is compromised by numerous magic numbers and insufficient comments explaining the reinforcement learning concepts. The implementation lacks proper modularity, with environment interaction, neural network definition, and training logic tightly coupled. Finally, the evaluation framework is limited, providing insufficient metrics to assess the agent's performance across different scenarios.

### **Enhancement Plans**

* Implement priority queue-based experience replay
* Refactor code to improve modularity
* Add detailed algorithm documentation
* Implement comprehensive testing metrics
* Optimize learning hyperparameters

To improve upon this artifact, I will add a priority queue-based experience replay system with higher priority given to higher TD-errors in order to significantly enhance learning efficiency. I will reorganize the code to better separate concerns into different modules for environment interaction, definition of the neural network, and training logic. Detailed documentation is included in order to describe reinforcement learning principles and implementation specifics. Robust testing metrics will be created to analyze the performance of the agent under different environmental settings. I will also fine-tune learning hyperparameters using systematic experimentation in order to improve convergence speed. These improvements illustrate skills in algorithm improvement and data structure choice according to course goals 3 and 4.

## **Category Three: Databases**

### **Existing Functionality**

* Global Rain Dashboard from CS 340
* Features:
  + Python/Dash web application with MongoDB
  + Basic filtering via dropdown menus
  + Data table for displaying results
  + Geolocation visualization
  + Basic outcome chart

This is an artifact consisting of an animal rescue organization web application dashboard used to select ideal dogs for training. Python and Dash have been integrated with MongoDB to generate an interactive dashboard for data analysis from shelters. There is an ability to filter animals through select boxes according to rescue types and desired breeds. Filtered results are presented in a responsive data table by the dashboard while animal locations are visualized using a geolocation chart and a simple visualisation of animal outcomes is presented. There is an ability through this tool for the organization to select and classify available canines based on specific training qualifications in an effective manner.

### **Code Analysis**

* **Database Queries**: Basic queries without advanced aggregation
* **Updates**: No real-time data updates
* **Query Efficiency**: Retrieves unnecessary data fields
* **Error Handling**: Minimal database error handling
* **Organization**: Scattered database access code

My database implementation holds a number of areas for improvement based on my analysis. The queries used now are simple and do not take advantage of MongoDB's advanced aggregation framework, meaning most data processing is being done in application code as opposed to database level. The dashboard shows static data and must be manually refreshed in order to see database changes. Retrieving entire documents when certain fields alone need to be used hampers query efficiency. Error handling is kept to a minimum for database activity and query failures and connection failures do not have an effective mechanism in place. Database access is interspersed throughout the application as opposed to all being contained in a single dedicated module, making maintainability lower.

### **Enhancement Plans**

* Implement MongoDB aggregation pipelines
* Add WebSocket for real-time updates
* Optimize queries and implement indexing
* Improve database error handling
* Create centralized database module

My improvement project will target using MongoDB's aggregation framework to execute advanced data analysis on the database level and minimize application-side computation. I'll implement WebSockets to automatically update dashboards in real-time as data changes in the database happen, removing manual refresh needs. Query optimization will be achieved by projecting required fields and using indexing over commonly queried fields for enhanced performance. I'll implement extensive error handling for database actions with easy-to-understand error messages. Last but not least, I'll implement a central database module to contain all MongoDB actions for better coding structure and maintainability. These improvements support learning outcome 3 and 4 through executing database solutions in an efficient manner that offers considerable user benefits.

## **Conclusion**

My planned enhancements will:

* Improve application architecture and security
* Optimize algorithm learning efficiency
* Enhance database integration and visualization

These enhancements highlight my skills in software design and software solution optimization in major areas of computer science in which I have worked. These enhancements show that I am capable of applying industry-standard best practices in software design, algorithmic optimization, and database design. These enhanced artifacts will be evidence of high-quality work that resolves security issues, performance constraints, and maintainability defects in the original designs. This code review reveals obvious, pragmatic areas for improvement that align with learning objectives while showcasing my advancement as a computer science practitioner.